Amendments to the Specification:

Please replace the paragraph beginning at page 2, line 12, with the following redlined paragraph:

This application relates to co-pending United States Patent Applications (a) Serial No. ~~__/___,____~~ 09/872,329, entitled METHOD AND SYSTEM FOR COMMUNICATING A REQUEST PACKET IN RESPONSE TO A STATE, (b) Serial No. ~~__/___,____~~ 09/872,539, entitled METHOD AND SYSTEM FOR COMMUNICATING AN INFORMATION PACKET AND IDENTIFYING A DATA STRUCTURE, (c) Serial No. ~~__/___,____~~ 09/873,019, entitled METHOD AND SYSTEM FOR INITIATING EXECUTION OF SOFTWARE IN RESPONSE TO A STATE, (d) Serial No. ~~__/___,____~~ 09/872,376, entitled METHOD AND SYSTEM FOR COMMUNICATING AN INFORMATION PACKET THROUGH MULTIPLE ROUTER DEVICES, (e) Serial No. ~~__/___,____~~ 09/872,372, entitled METHOD AND SYSTEM FOR ESTABLISHING A DATA STRUCTURE OF A CONNECTION WITH A CLIENT, (f) Serial No. ~~__/___,____~~ 09/872,332, entitled METHOD AND SYSTEM FOR IDENTIFYING A COMPUTING DEVICE IN RESPONSE TO A REQUEST PACKET, and (g) Serial No. ~~__/___,____~~ 09/872,081, entitled METHOD AND SYSTEM FOR EXECUTING PROTOCOL STACK INSTRUCTIONS TO FORM A PACKET FOR CAUSING A COMPUTING DEVICE TO PERFORM AN OPERATION. Each of these co-pending applications is filed concurrently herewith, names Mitchell T. Berg as inventor, is incorporated herein by reference in its entirety, and is assigned to the assignee of this application.

Please add between page 3, line 24 and page 3, line 25, the following moved paragraphs:

FIGURE 1a is a block diagram of a conventional system for processing information with a client computer system ("client") and server computer system ("server") that communicate (e.g. receive and output information) with one another through an Internet Protocol ("IP") global computer network (e.g. the Internet or an intranet). For clarity, FIGURE 1a shows

only a single client and a single server, although multiple clients and multiple servers are connected to the IP network. In FIGURE 1a, the client is a representative one of the multiple clients, and the server is a representative one of the multiple servers.

Conventionally, clients and servers communicate with one another through the IP network according to either the Transmission Control Protocol ("TCP") or User Datagram Protocol ("UDP"). In FIGURE 1a, a server makes its socket application (or "socket-based application") available through the IP network and waits for a client to establish a connection with the server through a specified IP address and TCP port (e.g. through a listening socket). For example, a server executing a World Wide Web application has a listening socket associated with an assigned 32-bit IP address on the standard TCP port 80 for a World Wide Web server application.

After accepting a connection from a requesting client, the server creates (or "establishes" or "forms") a client specific socket. The socket (created by the server) represents the server's connection for the sending (and receiving) information to (and from) the specific client. Conventionally, in response to creation of a socket, the server (with its operating system ("OS") kernel) allocates (or "establishes" or "forms") a data structure (of the connection with the client) to store client-to-server protocol specific connection information. This data structure is referred to as a socket connection endpoint (or "connection endpoint").

Information within the connection endpoint varies according to the type of connection established (e.g. TCP or UDP). For UDP and TCP types of connections, the connection endpoint information includes the client's and server's respective 32-bit IP addresses, the client application's and server application's respective 16-bit TCP connection ports, a pointer reference to a socket structure, and IP options such as Time to Live ("TTL") and Type of Service ("TOS"). Additionally, for a TCP type of connection, the connection endpoint information includes a group of send and receive sequence numbers (including start, current, and acknowledgement sequence numbers of the server and client) and variables for timing individual sent packets. In various embodiments, the connection endpoint information includes additional suitable information.

The client performs similar operations. With a socket layer (which manages sockets), the client (with a client application) creates a connection endpoint of a specified type (e.g. UDP or TCP) and attempts a connection to a server's listening socket. For example, with a conventional web browser (e.g. Netscape Navigator or Microsoft Internet Explorer), the client opens a TCP type of connection endpoint and attempts connection through an IP network to a web server through the web server's advertised IP address on the standard web service TCP port 80. After establishing a successful connection, the client and server are operable to send (and receive) information to (and from) one another through the associated socket connection.

With read and write calls to the socket layer, the client and server are operable to send and receive information at the application level. The client and server communicate with one another through IP packets sent through the IP network. Accordingly, before sending information from an application through the IP network (in response to a suitable connection endpoint), the computer system (e.g. client or server) encapsulates such information according to the IP protocol. Also, in response to receiving information from a network interface, the computer system (in response to a suitable connection endpoint) directs such information to an associated application.

As shown in FIGURE 1a, the client and server have respective protocol stacks, which process IP packets (sent and received) and manage connection endpoint information. With the protocol stack, the computer system (a) adds transport specific information before sending information to the network interface and (b) removes transport specific information before alerting an application of the receipt of information from the network interface. Conventionally, the protocol stack is part of the OS and executes in kernel mode.

The protocol stack includes a series of routines (e.g. software instructions) to process a packet in accordance with one or more network protocols such as HTTP, Ethernet, IP, TCP or UDP. In response to receiving a packet from the IP network, the network interface sends the packet through its associated device driver to the protocol stack's routines. For example, in response to receiving an IP packet, the computer system (with its protocol stack) verifies the IP packet according to the packet's checksum algorithm and then moves the packet up the protocol stack for additional processing in accordance with a network protocol.

At each level of the protocol stack processing, the computer system reads, processes and removes a header from the packet. At the end of protocol stack processing, the final result is information that the computer system stores in a destination socket queue. In response to information in the destination socket queue, the computer system (with its OS) initiates a software interrupt to the destination application, alerting the destination application that such information has been received.

For sending information through the network interface to the IP network, the computer system (with the socket application) outputs such information (which has been formed according to software instructions of the socket application) to the protocol stack along with a reference to a suitable connection endpoint. Then, the computer system (with the connection endpoint) moves the information down the protocol stack for additional processing in accordance with a network protocol. At various levels of the protocol stack processing, the computer system forms a packet by supplementing the information with TCP or UDP header information, IP header information, link layer header information (e.g. Ethernet), and calculation of packet checksums. After forming the packet, the computer system outputs the packet to a device driver output queue of the network interface.

Description of Conventional Flow Switch Architecture

FIGURE 1b is a block diagram of a conventional system for processing information with a group of servers ("server farm") and a client that communicate with one another through a global computer network with IP socket-based applications. In this example, a server farm (including *n* servers, where *n* is an integer number) stores the applications to be deployed. Server farms are useful for deploying software applications (e.g. web site application or Internet gaming site application) for use through a global computer network.

As shown in FIGURE 1b, each of the *n* servers is connected to a flow switch at egress ports of the flow switch. At an ingress port of the flow switch, it is coupled through a router to the IP network.

In the example of FIGURE 1b, a client connects to a server's application by connecting to the entire server farm through a single IP address. The IP address is associated

with the ingress port of the flow switch. Typically, the client obtains the IP address by sending a Uniform Resource Locator ("URL") to a Domain Name System ("DNS"). DNS is a set of special servers deployed on the IP network, with responsibility for translating a URL into an associated IP address. Alternatively, if a client has already received the IP address, the client is able to connect to the server farm without relying on the DNS.

All communications between a server (of the server farm) and a client are directed through the flow switch. The flow switch helps to balance client request loads on servers within the server farm ("server farm load-balancing") by selecting a server to handle a particular client's connection. Accordingly, the flow switch (a) maps packets from the flow switch's ingress port to the selected server through a suitable one of the flow switch's egress ports, (b) maps packets from the selected server to the particular client, and (c) performs various administrative operations. In processing a packet that is communicated between a server and a client, the conventional flow switch performs a range of operations, which may include network address translation ("NAT"), checksum calculation, and TCP sequence number rewriting ("TCP splicing").

Please delete the paragraphs beginning at page 7, line 25 and ending at page 11, line 3.

Please replace the paragraph beginning at page 12, line 1, with the following redlined paragraph:

Unlike the system of FIGURE 1b, in the system of FIGURE 2a, a client connects to a server farm application by obtaining and connecting to a server's IP address, instead of a flow switch's IP address. In the illustrative embodiments, the servers' respective IP addresses are advertised to clients in one of multiple possible ways. For example, according to a first technique, if multiple servers deploy a single application under a single URL, the DNS advertises IP addresses of those servers in a round-robin manner (e.g. one IP address at a time, alternating in a rotational manner). For example, if two servers deploy a web site application under a single

URL ~~(e.g. www.mysite.com),~~ (for example, hypothetical www.mysite.com), the DNS advertises the two servers' respective IP addresses (in association with the web site's URL) in round-robin manner.

Please replace the paragraph beginning at page 15, line 9, with the following redlined paragraph:

As shown in FIGURE 2d, servers 1 and 2 are grouped ("application 1 group") to deploy application 1, and servers 3, 4 and 5 ("application 2 group") are grouped to deploy application 2. For example, the server farm of FIGURE 2d is configurable to host two web sites ~~(e.g. www.firstsite.com~~ (for example, using hypothetical URLs www.firstsite.com and ~~www.secondsite.com)~~ www.secondsite.com) with a single IP network connection. Client requests to a first URL ~~(e.g. www.firstsite.com)~~ (for example, the URL www.firstsite.com) are processed by application 1 group, and client requests to a second URL ~~(e.g. www.secondsite.com)~~ (for example, the URL www.secondsite.com) are processed by application 2 group.

Please replace the paragraph beginning at page 15, line 15, with the following redlined paragraph:

For each web site, IP addresses are advertised by either the DNS round-robin approach or the redirector device round-robin approach, as discussed hereinabove in connection with FIGURE 2a. For example, IP addresses of servers 1 and 2 are associated with the first URL ~~(www.firstsite.com),~~ (for example, hypothetical URL www.firstsite.com), and such IP addresses can be advertised in round-robin manner. Similarly, IP addresses of servers 3, 4 and 5 are associated with the second URL ~~(www.secondsite.com),~~ (hypothetical URL www.secondsite.com), and such IP addresses can be advertised in round-robin manner.

Please replace the paragraph beginning at page 17, line 8, with the following redlined paragraph:

In response to such a match, the network processor is operable to perform an action on the packet (e.g. send the packet to the protocol stack) in response to software instructions stored in the iNIC's memory (e.g. SRAM/SDRAM). In the illustrative embodiments, the network processor is a commercially available processor, such as Intel's IXP1200 processor ~~(available from www.intel.com)~~ or Motorola's C-5 Digital Communications ~~processor (available from www.motorola.com).~~ processor.

Please replace the paragraph beginning at page 55, line 29, with the following redlined paragraph:

Advantageously, the illustrative embodiments are compatible with conventional techniques in development of applications (and associated software component objects) deployed within a server farm for IP networks. A conventional development cycle involves the development of an application with reusable software objects (or component objects) that are deployed in a middleware component model, such as the development of an application process that calls service objects deployed in a middleware component model. Commercially available embodiments of middleware component models include Microsoft's Transaction Server ~~(available from www.microsoft.com)~~ and BEA's WebLogic ~~Server (available from www.BEA.com).~~ Server.